Tehnici Web
CURSUL 10

Semestrul I, 2024-2025

Carmen Chirita

OBIECTE

- un obiect este o colecție de perechi proprietate-valoare
- daca valoarea este o funcție, atunci proprietatea se numește metodă

```
var ob = {prop1: val1, prop2: val2, ... , propn: valn};
```

- accesarea proprietăților:

```
ob.prop1;  // val1
ob["prop1"]; // val1
```

PROTOTIPURI

- prototipul unui obiect este desemnat prin Object.prototype

- orice obiect moștenește proprietățile obiectului prototip (prototypal inheritance)

- obiectele care au același prototip formează o clasă

- toate obiectele sunt descendenți ai obiectului generic Object

Object.getPrototypeOf() // prototipul obiectului specificat

Crearea obiectelor

- prin **object literal**

- proprietățile, metodele, împreună cu valorile lor sunt enumerate între acolade

- se creează un singur obiect

```
var reteta = {nume: "Pateuri", durata: 40, categorie:"Aperitiv" }
```

Crearea obiectelor

- cu ajutorul **obiectului generic Object**

- se apeleaza constructorul new Object() și se adauga apoi proprietatile și metodele

- se creează un singur obiect

```
var reteta = new Object();
 reteta.nume = "Pateuri";
 reteta.durata = 40;
 reteta.categorie = "Aperitiv";
```

Crearea obiectelor

- cu ajutorul **unui constructor de obiecte**

- se defineste o funcție constructor(parametrii) care apoi va fi apelata cu new constructor(parametrii) pentru fiecare obiect care va fi creat

```
function reteta(n, d, c){
  this.nume = n;
  this.durata = d;
  this.categorie = c; }

var r1 = new reteta("Pateuri", 40, "Aperitiv");
var r2 = new reteta("Supa", 120, "Fel principal");
var r3 = new reteta("Tort", 100, "Desert");
```

Crearea obiectelor

- cu metoda **Object.create()**

- Object.create(ob)

- creează un nou obiect, folosind un obiect existent *ob* ca prototip al obiectului nou creat

Object.create(ob)

```
var  interval = {
    mx: 2,
    my: 4,
    apartine: function(z){
        return  (z <=  this.my) &&  (z >= this.mx);
     }
  };  //clasa
```

```
 var obi = Object.create(interval);  // obiect din clasa interval
  obi.mx =5;
  obi.my=7;  //obi suprascrie proprietatile prototipului

 var  intervalD = Object.create(interval);
 intervalD.apartine =  function(z){
              return  (z <  this.my) &&  (z > this.mx);
              };  //subclasa

var obid = Object.create(intervalD);    // obiect din clasa intervalD
 obid.mx = 5;
 obid.my=10;

interval.valid  = function(){return (this.my  >=  this.mx);};
intervalD.vid  = function(){return (this.mx == this.my);};

alert(obid.valid()); alert(obid.vid());
```

Cuvantul cheie this

- în interiorul unui constructor sau al unei metode asociate unui obiect, *this* se referă la obiectul curent

- într-o funcție folosită ca event handler, *this* se referă la elementul pentru care este definit listenerul

- altfel, *this* se referă la obiectul window

Crearea obiectelor folosind o functie constructor si new

```
function Interval(x, y) {
      this.mx= x;
      this.my= y; }  // clasa

 Interval.prototype.apartine = function(z){
           return  (z <=  this.my) &&  (z >= this.mx);}

 // metoda adaugata în prototipul obiectelor create cu functia
 constructor


var obi = new Interval(1,4); // obiect din clasa Interval


Interval.prototype.valid  = function(){return (this.my  >=  this.mx);};

alert(obi.valid()); //true
```

Crearea obiectelor folosind o functie constructor si new

Definirea subclaselor

```
function Interval(x, y) {
      this.mx= x;  this.my= y; }  // clasa

 Interval.prototype.apartine = function(z){
          return  (z <=  this.my) &&  (z >= this.mx);}
```

```
function IntervalD(x,y) { Interval.call(this,x,y);  } //this este obiectul care
                                                       se construieste

IntervalD.prototype = Object.create(Interval.prototype); //am schimbat prototipul
                                                          obiectelor create cu IntervalD
IntervalD.prototype.constructor = IntervalD; //restaurez proprietatea constructor
IntervalD.prototype.apartine = function(z){
                                       return  (z <  this.my) &&  (z > this.mx);};

var obid = new IntervalD(5,10);
```

Se pot adauga proprietăți noi: obid.mz = 1;

Crearea obiectelor folosind o functie constructor si new

Definirea subclaselor

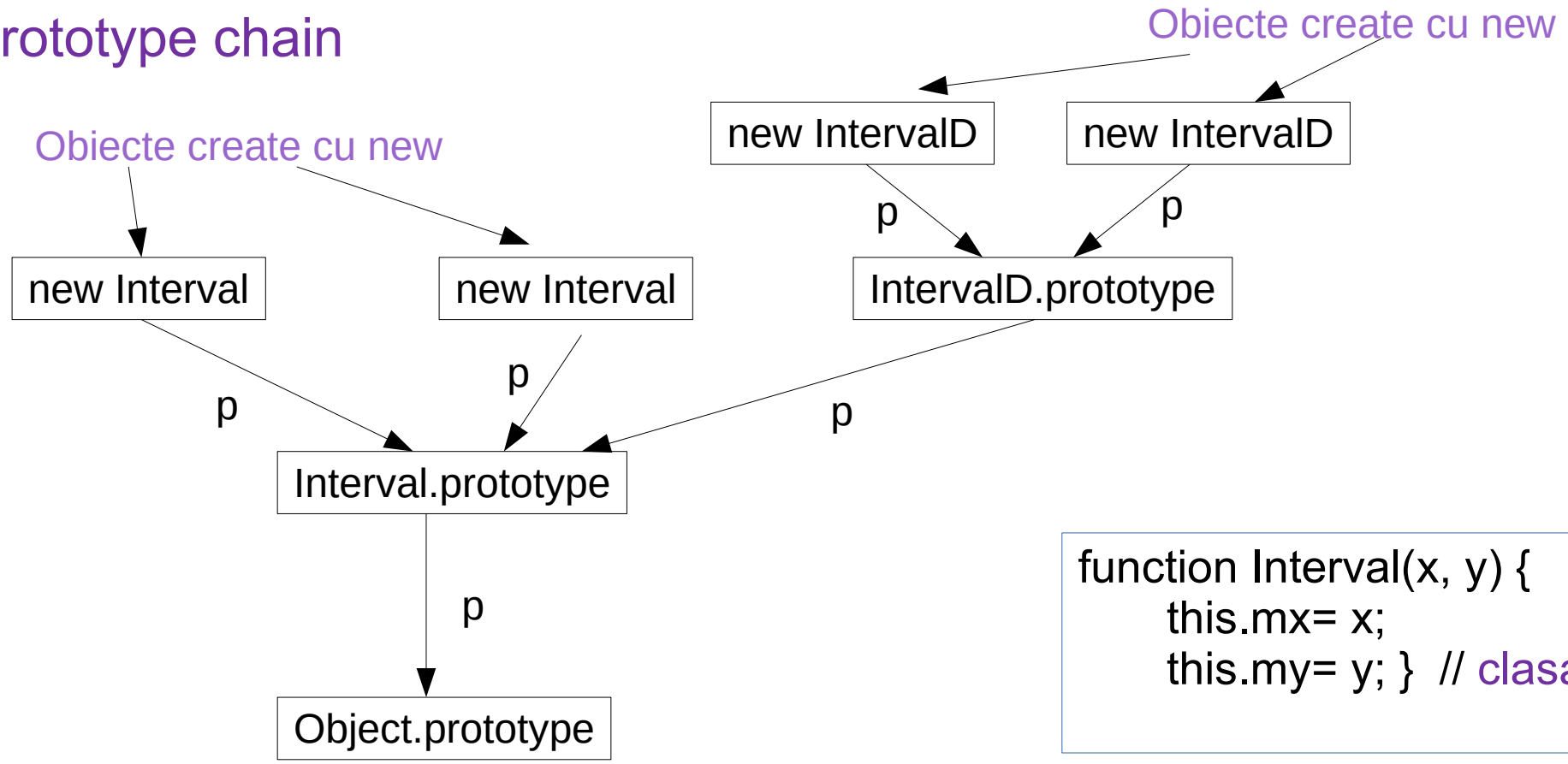
```
function Interval(x, y) {
      this.mx= x;  this.my= y; }  // clasa

 Interval.prototype.apartine = function(z){
          return  (z <=  this.my) &&  (z >= this.mx);}
Interval.prototype.valid  = function(){return (this.my  >=  this.mx);};
```

```
function IntervalD(x,y) {    Interval.call(this,x,y);  }

IntervalD.prototype = Object.create(Interval.prototype);
IntervalD.prototype.constructor = IntervalD;
IntervalD.prototype.apartine = function(z){
                                        return  (z <  this.my) &&  (z > this.mx);};
IntervalD.prototype.valid = function(){
   return  (Interval.prototype.valid.call(this) && (this.mx != this.my));
}
```

Prototype chain




```
function Interval(x, y) {
      this.mx= x;
      this.my= y; }  // clasa
```

```
function IntervalD(x,y) {Interval.call(this,x,y);  }

IntervalD.prototype = Object.create(Interval.prototype);
//subclasa
```

Expresii regulate
(RegExp)

sunt folosite în numeroase limbaje de programare pentru

validări, parsări, căutări și înlocuiri în texte

Ce sunt expresiile regulate?

- sunt siruri speciale de caractere care reprezintă un sablon de căutare
- sunt un tip de obiect

La ce folosesc?

- sunt folosite în special pentru validarea datelor
- pentru cautare/inlocuire/potrivire a unui sir într-un anumit format în cadrul altui sir

Crearea unei expresii regulate

- folosind constructorul obiectului RegExp
  new RegExp(”sir”,”modificatori”)
- prin valoare literala, sirul fiind incadrat intre caracterul „/”
  /sir/modificatori

Modificatorii sunt utilizati pentru a efectua cautari globale si care nu tin cont de majuscule si minuscule

g - căutare globala
i - nu face diferența intre literele mici și mari
m - forma sirului se aplica pentru fiecare linie
d - genereaza un vector de indici dacă se găsesc mai multe subsiruri care se potrivesc cu expresia regulata

```
let e1 = new RegExp(”abc”);
let e2= /abc/;
```

Metacaractere

- sunt caractere speciale cu funcționalitate predefinită

. - orice caracter (exceptând newline-ul)

^ - inceputul șirului

$ - sfârșitul șirului

| - sau (alternativă)

( ) - grupare

[ ] – clase de caractere

{ } – specifică numărul de apariții

Clase de caractere

- definesc grupuri de caractere care pot apărea într-o anumită poziție din text:

[abc] - oricare dintre caracterele a, b sau c

[^abc] - oricare alt caracter, cu excepția celor din clasă

[a-z] - orice literă mică din alfabet

[A-Z] - orice literă mare din alfabet

[0-9] - orice cifră

[a-zA-Z0-9_] - orice caracter alfanumeric și underscore (_)

Clase predefinite

\d - orice cifră ([0-9])

\D - orice caracter care nu este o cifră ([^0-9])

\w - orice caracter alfanumeric sau _ ([a-zA-Z0-9_])

\W - orice caracter care nu este alfanumeric ([^a-zA-Z0-9_])

\s - orice spațiu alb (inclusiv tab și newline)

\S - orice caracter care nu este spațiu alb

Cuantificatori: specifică de câte ori trebuie să apară un element

a***** - zero sau mai multe apariții ale caracterului sau grupului anterior

a**+** - una sau mai multe apariții ale caracterului sau grupului

a**?** - zero sau o apariție ale caracterului sau grupului

a**{n}** - exact n apariții ale caracterului sau grupului

a**{n,}** - cel puțin n apariții ale caracterului sau grupului

a**{n,m}** - intre n si m apariții ale caracterului sau grupului

a(**?=b)** - recunoaște a dacă este urmat imediat de b (fără a-l consuma pe b)

Metoda test()

- executa o căutare a expresiei regulate într-un sir și întoarce true sau false

```
/hello/.test("Hello world"); //false
/hello/i.test("Hello world"); //true
/[0123456789]/.test("in 1992"); //true

let dateTime = /\d\d-\d\d-\d\d\d\d \d\d:\d\d/;
dateTime.test("01-04-2024 15:20");  //true
dateTime.test("29-aprilie-2024 15:20"); //false

/ab[a-f][0-25-9]/.test("abc7"); //true
/ab[a-f][0-25-9]/.test("abc34"); //false

/[Hh]alt[A-Z]/.test("HaltA");  //true
/[Hh]alt[A-Z]/.test("HALTA"); //false
```

Metoda test() - exemple

```
let nume1 = /^[A-Z][a-z]{2,}$/;

nume1.test("Ana");   //true
nume1.test("Maria"); //true
nume1.test("Lu”);  //false

let nume2 =  /^[A-Z][a-z]{2,}(-[A-Z][a-z]{2,})?$/;

nume2.test("Ana-Maria"); //true
nume.2.test("Sofia"); //true
nume2.test("Ana-Maria-Sofia"); //false

let nume3 =  /^[A-Z][a-z]{2,}(-[A-Z][a-z]{2,})*$/;
nume3.test("Ana-Maria-Sofia"); //true
```

Metoda test() - exemple

```
let user =  /^[a-zA-Z][a-zA-Z0-9_]{2,}$/;

user.test("abc123”); //true
user.test("12abc"); //false
user.test("ab_24"); //true

let parola = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[?!$%&#@]).{8,}$/;

parola.test("Abcdef1$");  //true
parola.test("abcdefg1"); //false
parola.test("12ab_C!x"); //true
```

Metoda test() - exemple

```
let telefon = /^\+40\d{9}$/;

telefon.test("+40111222333"); //true
telefon.test("40111222333"); //false

let email = /^[a-zA-Z0-9._%+-]+@([a-zA-Z0-9-]+\.)+[a-zA-Z]{2,}$/;

email.test("utilizator@gmail.com"); // true
email.test("nume.prenume@yahoo.co.uk"); //true
email.test("user_123@domeniu.org"); //true


email.test("user@@domeniu.com"); //false
email.test("user.domeniu.com"); //false
email.test("user@.com"); //false
```